INTEGRACIÓN DE LAS PANTALLAS DITEL SERIE DINOS-DMG EN SISTEMAS OPERATIVOS WINDOWS 32 Y 64 BITS

ÍNDICE

Requisitos del sistema……..... 3

Funcionamiento general 4

Propiedades..... 6

Métodos 14

Detalle de la utilización de las órdenes……... 15

Listado de ordenes..... 20

Protocolos 20

Constantes para los programas ……. 20

Lista de las Propiedades, Métodos.y Eventos..... 22

Utilización de MPCOMM.OCX dentro de .NET…….. 25

Códigos de Programación……... 26

1. Requisitos del sistema (compatibilidad)

MPComm es un componente ActiveX. Este componente puede ser utilizado en sistemas operativos Windows 32 y 64 bits.

La tecnología ActiveX se puede utilizar con la mayoría de sistemas actuales de desarrollo por lo que MPComm resulta una solución ideal para la distribución de este tipo de soluciones.

Las pruebas se han realizado utilizando C# y VisualBasic 6.0.

Para el correcto funcionamiento de MPComm es necesario establecer las tres propiedades referentes a la licencia.

LicProduct	Nombre del Producto	MPCOMM
LicName	Nombre de la Licencia	PIXELMP
LicKey	Código de licencia	**“431A2CF1683AAFAC”**

2. Funcionamiento General

MPComm es un ActiveX que puede ser incrustado en cualquier aplicación 32Bits que haya sido desarrollada con los últimos sistemas operativos compatibles con esta tecnología.

El control dispone de :

Propiedades : Nos permiten configurar el control para abrir puertos serie RS232, realizar conexiones por TCP/IP, establecer y recuperar la información durante los procesos de comunicación entre el control y los diferentes dispositivos que se desee controlar.

Eventos : Los eventos nos informarán en tiempo de proceso de la actividad del control con los diferentes dispositivos que desee realizar.

Métodos : Los métodos nos permitirán ejecutar funciones propias del control.

Para utilizar el control primero hay que inicializar el componente MPComm después abrir el port serie o establecer una conexión TCP/IP. Después de cada control se recomienda cerrar la comunicación.

Ejemplo : inicializar MPComm (C#)

```
axMPComm1.LicProduct = "MPCOMM";            //Nombre del producto
axMPComm1.LicName = "PIXELMP";              //Nombre de la licencia
axMPComm1.LicKey = "431A2CF1683AAFAC";     //Codigo de la licencia
axMPComm1.Terminal2 = 8;                    //
axMPComm1.Terminal1 = 4;                    //
axMPComm1.DebugMode = false;                //Modo de depuración
```

Ejemplo : abrir Puerto Serie (VB)

```
MPComm1.Port = “COM1”   // Puerto
MPComm1.Speed = “38400” // Velocidad
MPComm1.Protocolo = 0   // Protocolo.   0 -> RS232/485, 1 -> TCP/IP
MPComm1.RS485 = false   // Tipo de linea. False ->RS232, True -> RS485
MPComm1.OpenComm        // Abrir puerto
```

Ejemplo : abrir Puerto Serie (C#)

```
axMPComm1.Port = "COM1:";
axMPComm1.Speed = "9600";
axMPComm1.Protocolo = 0;
axMPComm1.RS485 = false;
axMPComm1.OpenComm();
```

Ejemplo : establecer comunicación TCP / IP (VB)

```
MPComm1.HostAdress = gl_iphost$  //IP Visualizador
MPComm1.HostPort = gl_ipport$  // TCP port
MPComm1.Usuario = gl_ipusr$      //
MPComm1.Password = gl_ippass$
MPComm1.Protocolo = 1
MPComm1.RS485 = False
MPComm1.OpenComm
```

Ejemplo : establecer communicación TCP / IP (C#)

```
axMPComm1.TipoServidor = 1;    //  Valor por defecto
axMPComm1.HostPort = 53;       //  Puerto TCP.Valor por defecto
axMPComm1.Usuario = "admin";         // Valor por defecto
axMPComm1.Password = "security";    //  Valeur par défaut
axMPComm1.HostAdress = "192.168.1.44"; //Dirección IP visualizador
axMPComm1.Protocolo = 1;              //Protocole.   0 -> RS232/485, 1 -> TCP/IP
axMPComm1.OpenComm();              // Abrir puerto
```

Después de la apertura de una comunicación se puede comunicar con los diferentes aparatos. En el ejemplo siguiente, transmitimos al visualizador un programa de test llamado «**Test1**» en modo **INMEDIATO** utilizando la propiedad **Orden** (ver página 18) a la cual se asigna el valor 39 (0x27) que corresponde a la orden **FASTEXEC** (ejecución inmediata de un mensaje). La lista detallada de los comandos que se pueden utilizar con la propriedad **Orden** se encuentra en la página 20.

Ejemplo : Enviar un mensaje al visualizador (VB)

```
MSG$ = Chr$(MPMOD_INMEDIATO) + "Test1"   // mensaje a enviar (String)
MPComm1.BufferTX = MSG$                  // Buffer = mensaje
MPComm1.Orden = 0x27        // comando FASTEXE (Enviar inmediatamente)
numero_error = Str$( MPComm1.ErrorNx     // résultado de la  comunicación
Error$ = MPComm1.ErrorTx                 // Texto Error
```

Ejemplo : Enviar un mensaje al visualizador (C#)

```
string MSG = "Test1";                // mensaje a enviar (String)
axMPComm1.Pantalla = 1;              // Visualizador con ID nº 1
axMPComm1.BufferTX = MSG;            // Buffer = mensaje
axMPComm1.Orden = 0x27;      // comando FASTEXE (Enviar inmediatamente)
Thread.Sleep(200);                   // Temporización de 200ms
String s = axMPComm1.        // Respuesta del visualizador (si retorna valor)
int error = axMPComm1.ErrorNx;       // Resultado de la comunicación
```

Es muy importante diferenciar los comandos de comunicación (FASTEXEC, EXECUTE, SETHORA) de los comandos de programación (INMEDIATO,SUBE,VELMODO). Los comandos de comunicación nos permiten enviar y recibir las informaciones de un visualizador. Los comandos de programación nos permiten establecer una secuencia de instrucciones a ejecutar sobre el visualizador una vez se ha transmitido el programa.

Finalmemte, hay que cerrar la conexión para poner fin al proceso de comunicación.

Ejemplo : Terminar la comunication (C#)

```
MPComm1.CloseComm   //Cierra el puerto serie o la conexion TCP/IP
```

3. Propiedades

BufferRx

La propiedad BufferRx nos permitirá obtener el resultado de todas aquellas ordenes que devuelvan datos como por ejemplo ; solicitar la fecha de una pantalla, obtener el programa en ejecución, el directorio

Tipo	Lectura / Escritura	Formato	Rango
String	Si / No	(Según Orden)	-

```
MPComm1.Pantalla=1
MPComm1.Orden = 0x0B  // comando GETHORA (Pide la ho
ra del visualizador)
```

BufferTx

Algunos comandos de protocolo no realizan una función especifica, pero permiten transmitir informaciones al display. La información que debe ser transmitida debe almacenarse en esta propiedad con el formato adecuado antes de ejecutar el comando.

Tipo	Lectura / Escritura	Formato	Rango
String	No / Si	(Según Orden)	-

```
MPComm1.Pantalla = 1
MPComm1.BufferTx =”01-01-99 12:00:00”
MPComm1.Orden = 0x0A  // comando SETHORA (Pone en hora
                         el reloj del visualizador
```

CFGNumPan

UNIQUEMENT POUR RS232 .

Define el número de visualizador (ID) en la configuración interna. Este campo tiene un rango de 0 à 253.

Tipo	Lectura / Éscritura	Formato	Rango
Entero	Si/Si	-	0 - No utilizado 1–253 Recomandado 254 - Maestro 255 - Difusión

```
MPComm1.CFGNumPan = 1
```

CFGBaudsRS232

Unicamente para RS232

Indica la velocidad del puerto RS232 de placa de control en la configuración interna. Una vez modifique este parámetro en la configuración deberá cerrar el puerto de comunicaciones cambiar su configuración y abrirlo de nuevo.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si		0-1200 1-2400 2-4800 3-9600

```
MPComm1.CFGBaudRS232 = 3
MPComm1.Orden = 0x0B  // comando GETHORA (Pide la hora del visualizador)
Hora_Visualizador$ = MPComm1.BufferRx
```

CFGPassword

Esta propiedad permite poner, modificar o eliminar el password en la placa de control. Si establece esta propiedad con ocho espacios en blanco y transmite la configuración esta quedará desactivada. Para poder asegurarse de los cambios en el password han sido efectuados deberá reiniciar la pantalla para que sea obligatorio mandar la contraseña.

Tipo	Lectura / Escritura	Formato	Rango
String	Si/Si	8A	-

```
MPComm1.CFGPassword = “CSECRETO”
```

CFGBaudsRS485

Especifica la velocidad de la configuración interna del port RS485. Si la carta de control no dispone del port RS485 , debe ser igual a la velocidad del puerto RS232. En caso contrario pueden tener velocidades diferentes.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si		0-1200 1-2400 2-4800 3-9600

```
MPComm1.CFGBaudsRS485 = 1
```

CFGOffsetTemp

Esta propiedad determina en la configuración de la carta de control que valor deberá incrementar o decrementar al mostrar la temperatura. El campo no afecta en caso de notener sensor de temperatura.

Tipo	Lectura / Escritura	Formato	Rango
Signed	Si / Si	-	-

MPComm1.CFGOffsetTemp = -1

CFGFlPowerOnReset

Indica a la placa de control si al iniciarse debe realizar un reset de forma automática e inicializar la memoria de pantalla.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si		0-Si 1-No

MPComm1.CFGFlPowerOnReset = 1

CFGFlNoCls

El flag para ClsStop es de gran utilidad ya que permite configurar si al realizar un STOP en la pantalla queremos que el mensaje de la misma sea borrado. En algunos casos la contínua llegada de información a la pantalla provoca un parpadeo no deseado que puede ser evitado con esta propiedad.

NOTA :Esta misma acciónpuede realizarse simplemente enviando un mensaje vacio antes de parar el indicador.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si		0-Si 1-No

MPComm1.CFGFlNoCls = 1

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	0-255

MPComm1.CFGVelModoDef = 1

DebugMode

Esta propiedad ofrece la posibilidad de generar un registro en la ubicación C: \ MPCOMM.DBG con todas las comunicaciones realizadas indicando la dirección y los milisegundos entre comunicaciones. Esta propiedad es de gran utilidad para depurar el producto final.

Tipo	Lectura / Escritura	Formato	Rango
Bool	Si / Si	-	True-False

```
2768895 --> (1)16 15 00 01 13 42 42 56 00 B0 40 00 00 04 C9 00 00 30 39 3F 03
2768970 <-- ACK:00
2872454 --> (1)16 0F 00 01 06 49 4E 46 4F 42 4F 4C 53 88 02
2872595 <-- ACK:00
3788235 --- Puerto serie Cerrado
```

ErrorNx

Utilice la propiedad ErrorNx después de cada comunicación para comprobar el resultado. En condiciones normales si la comunicación ha sido satisfactoria la propiedad tendrá su valor a 0, en caso contrario mostrará un mensaje de error (ErrorTx)

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / No	-	0-1050

```
Dim Numero_error AS Integer

MPComm1.Pantalla = 1
MPComm1.BufferTx =chr$(MPMODO_INMEDIATO)+"PRUEBA DE TEXTO"
MPComm1.Orden = MPCMD_FASTEXEC

Numero_error = MPComm1.errornx
```

ErrorTx

La propiedad ErrorTx es muy semejante a ErrorNx con una peculiaridad muy importante. El resultado del error se devuelve en forma de string. Resulta de gran utilidad durante el periodo de depuración de un programa poder saber qué error se está produciendo sin necesidad de recorrer a una tabla de conversión. Esta propiedad se actualiza al final de cada conversión.

Tipo	Lectura / Escritura	Formato	Rango
String	Si / No	-	-

```
Dim Numero_error AS Integer
Dim Texto_error AS String
MPComm1.Pantalla = 1
MPComm1.BufferTx =chr$(MPMODO_INMEDIATO)+"PRUEBA DE TEXTO"
MPComm1.Orden = MP_FASTEXEC
if MPComm1.errornx<>0 then
   Texto_error=MPComm1.ErrorTx
    Msgbox (Texto_error)
end if
```

EsperaACK

Especifica el tiempo en milisegundos que debe esperar el ACK de un dispositivo antes de dar error TimeOut

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	(IP)	1-10000

MPComm1.EsperaACK = 3000 (3 segundos)

HostAdress

Propiedad encargada de almacenar la dirección IP del visualizador.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	(IP)	255.255.255.255

MPComm1.HostAdress="127.0.0.1"

HostPort

Hostport funciona conjuntamente con HostAdress para determinar el puerto TCP del servidor al que debe conectarse. Habitualmente la dirección IP y el numero de puerto de un servidor no depende de nuestro software, sino que nos deberán facilitar dicha información.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	1-65535

MPComm1.HostAdress="127.0.0.1"
MPComm1.Hostport="1001"

Orden

Utilizaremos esta propiedad para indicar a MPComm qué orden deseamos ejecutar en la pantalla. Dependiendo de la orden el control esperará o no respuesta y almacenará en la propiedad **BufferRx** los datos recibidos, o bien utilizará la propiedad **BufferTx** para transmitir las informaciones.

Tipo	Lectura / Escritura	Formato	Rango
Entero	No / Si	-	0-255

```
MPComm1.Pantalla = 1
MPComm1.BufferTx =chr$(MPMODO_INMEDIATO)+"PRUEBA DE TEXTO"
MPComm1.Orden = MP_FASTEXEC
MPComm1.Pantalla = 1
MPComm1.Orden = MP_GETDIR
Directorio$= MPComm1.BufferRx
```

Pantalla

La propiedad Pantalla define la dirección ID del visualizador con el que queremos realizar el proceso de comunicación. El visualizador 255 esta reservado a la difusión.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	0-255

MPComm1.Pantalla = 1

PassWord

La propiedad PassWord nos permite establecer la palabra de paso a utilizar en el transcurso de una comunicación con protocolo TCP / IP. Esta propiedad funciona conjuntamente con la propiedad Usuario.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	0-10

MPComm1.PassWord =”ULZ2300”

Port

La propiedad Port almacena el numero de puerto RS232 que se desea abrir y utilizar para realizar las transmisiones. Esta propiedad se utiliza cuando se trabaja con el protocolo RS232.

Tipo	Lectura / Escritura	Formato	Rango
Entero	No / Si	-	COM1 - COM4

MPComm1.Port=”COM1”

Protocolo

Esta propiedad tiene gran importancia ya que permite conmutar el protocolo o medio de transmisión entre RS232 y TCP/IP. Esto significa que solo cambiando esta propiedad el control podrá dirigir las comunicaciones en uno de estos dos modos.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	0-1

MPComm1.Protocolo = MPPROT_RS232

0 Protocolo RS232
1 Protocolo TCP/IP

Reintentos

Esta propiedad nos permite establecer el numero de reintentos que deseamos que el control realice de forma automática antes de considerar un posible error. En la mayoria de los casos se suelen realizar tres intentos, aunque en determinados momentos puede ser mas recomendable utilizar uno.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	0-10

```
MPComm1.Reintentos = 1
```

Speed

La propiedad Speed se utiliza para configurar la velocidad de comunicación entre el PC y la pantalla o red de pantallas en RS485. Las velocidades disponibles son

Tipo	Lectura / Escritura	Formato	Rango
String	Si / Si	-	1200 2400 9600 19200 38400

```
MPComm1.Speed = ”9600”
MPComm1.Port    = ”COM1”
MPComm1.OpenComm
```

Usuario

Establece el nombre del usuario a utilizar para realizar una conexión con el visualizador. Esta propiedad solo se utilizará para si se utiliza el protocolo TCP/IP y su utilización será conjunta a la propiedad Password

Tipo	Lectura / Escritura	Formato	Rango
String	Si / Si	-	-

```
MPComm1.Usuario  = ”Administrador”
```

VariableIDX

Establece el código de variable que se desea modificar. Las variables se identifican de forma numérica, siendo la variable A el número 0..

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	1-28

```
MPCommObj.VariableIdx = 2  ‘ Variable C
MPCommObj.VariableValor = Val(Text1(0))
MPCommObj.Orden = MPCMD_PUTVARS
```

VariableValor

Establece el valor de la variable que se seleccionó con VariableIDx. Este valor se introduce de forma entera. Se puede especificar el formato de salida indicando el número de dígitos enteros y decimales antes de la variable:
Ejemplo: 5.2A 5 dígitos con 2 decimales para la variable A.

Tipo	Lectura / Escritura	Formato	Rango
Entero	Si / Si	-	1-65535

```
MPCommObj.VariableIdx = 2  ‘ Variable C
MPCommObj.VariableValor = 123
MPCommObj.Orden = MPCMD_PUTVARS
```

Ejemplo de programa con variables. Las variables pueden venir definidas con:

```
txt$ = Chr$(MPDAT_SYNC)
txt$ = txt$ + Chr$(MPDAT_LINIA) + "1"
txt$ = txt$ + Chr$(MPMOD_INMEDIATO) + " VAR A=" + Chr$(MPTIM_VARIABLE) + "A"
txt$ = txt$ + Chr$(MPDAT_LINIA) + "2"
txt$ = txt$ + Chr$(MPMOD_INMEDIATO) + "VAR B=" + Chr$(MPTIM_VARIABLE) + "5.2B"
txt$ = txt$ + Chr$(MPDAT_LINIA) + "3"
txt$ = txt$ + Chr$(MPMOD_INMEDIATO) + "VAR C=" + Chr$(MPTIM_VARIABLE) + "5.0C"
txt$ = txt$ + Chr$(MPDAT_SYNC)

MPCommObj.OpenComm
If MPCommObj.ErrorNx Then Exit Sub


MPCommObj.Pantalla = 1
MPCommObj.BufferTX = Chr$(MPMOD_INMEDIATO) & txt$
MPCommObj.Orden = MPCMD_FASTEXEC
```

---

4. Métodos

OpenComm

Abre el puerto serie o una conexión TCP/IP utilizando las propiedades referentes al protocolo, velocidad y puerto serie. Este método no utiliza parámetros. El resultado del método se moverá a la propiedad ErrorNx y ErrorTx.

Parámetros
-

```
MPComm1.Speed = ”9600”
MPComm1.Port    = ”COM1”
MPComm1.OpenComm
```

```
MPComm1.HostAdress = “195.106.100.2”
MPComm1.HostPort = “1001”
MPComm1.Usuario = “Usuario”
MPComm1.Password = “Clave”
MPComm1.Protocolo = 1
MPComm1.OpenComm
```

CloseComm

Cierra el puerto serie o una conexión TCP/IP. Este método no utiliza parámetros. El resultado del método se moverá a la propiedad ErrorNx y ErrorTx.

Parámetros
-

```
MPComm1.CloseComm
```

---

5. Detalles de utilización de los comandos.

```
MPCMD_RESET = 1          ' Inicializa la pantalla

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.Orden = MPCMD_RESET
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm

MPCMD_WDRESET = 2        ' Resetea la pantalla
MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.Orden = MPCMD_WDRESET
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm

MPCMD_STOP = 3           ' Para la ejecución de la pantalla

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.Orden = MPCMD_STOP
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_BORRAR = 5        ' Borra un programa
MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.BufferTX = "RELOJ"
MPComm1.Orden = MPCMD_BORRAR
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_GETPROG = 9       ' Solicita un programa existente
// Por ejemplo para recuperar el código de un programa hecho con el software de edición Dynamic Plus. El código recuperado puede reutilizarse directamente con el comando MPCMD_FASTEXEC

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.BufferTX = "PRUEBA"
MPComm1.Orden = MPCMD_GETPROG
Programa$ = MPComm1.BufferRx
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_SETHORA = 10      ' Establece la hora de la pantalla
MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.BufferTX = "01-01-05 13:30:16"
MPComm1.Orden = MPCMD_SETHORA
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_GETHORA = 11      ' Solicita la hora de la pantalla

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.Orden = MPCMD_GETHORA
iret$ = MPComm1.BufferRx

        TxtFecha(2) = Asc(Mid(iret$, 1, 1))
        TxtFecha(1) = Asc(Mid(iret$, 2, 1))
        TxtFecha(0) = Asc(Mid(iret$, 3, 1))

        TxtHora(0) = Asc(Mid(iret$, 4, 1))
        TxtHora(1) = Asc(Mid(iret$, 5, 1))
        TxtHora(2) = Asc(Mid(iret$, 6, 1))

ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_PASSWORD = 14     ' Envia el password a la pantalla

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.BufferTX = "SECRET"
MPComm1.Orden = MPCMD_PASSWORD
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_SETPASSW = 15     ' Establece Nuevo password
MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.BufferTX = "SECRET"
MPComm1.Orden = MPCMD_SETPASSW
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_GETVER = 18       ' Solicita versión de software
MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.Orden = MPCMD_GETVER
respuesta$ = main.MPComm1.BufferRx
version = Val("&h" + Mid(respuesta$, 1, 2))
hardhare$ = Mid(respuesta$, 4)
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_NGETDIR = 30      ' Solicita el directorio a la pantalla
MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Orden = MPCMD_NGETDIR
Directorio$   =MPComm1.BufferRx
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

```
MPCMD_NEJECUTAR = 31    ' Ejecuta un programa en la pantalla

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.BufferTX = "MPTEST"
MPComm1.Orden = MPCMD_NEJECUTAR
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm


MPCMD_FASTEXEC = 39     ' Ejecución instantánea de un programa

MPComm1.DebugMode = false
MPComm1.Terminal2 = 8
MPComm1.Terminal1 = 4
MPComm1.Port = "COM1"
MPComm1.Speed = "9600"
MPComm1.Protocolo = 0
MPComm1.RS485 = False
MPComm1.OpenComm
MPComm1.Pantalla = 1
MPComm1.BufferTX = Chr$(MPMOD_INMEDIATO) +
"Prueba"
MPComm1.Orden = MPCMD_FASTEXEC
ErrorNx  = Str$(MPComm1.ErrorNx)
ErrorTx$ = MPComm1.ErrorTx
MPComm1.CloseComm
```

6. Listado de ordenes.

A definir en la aplicación como variables globales en format « string ».

```
Ejemplo (C#) :
string MPCMD_FASTEXEC = char (0x27) +"";
axMPComm1.Orden = MPCMD_FASTEXEC;
```

```
Global Const MPCMD_RESET = 1(0x01)' Inicializa la pantalla
Global Const MPCMD_WDRESET = 2(0x02)' Resetea la pantalla
Global Const MPCMD_STOP = 3(0x03)' Para la ejecución de la pantalla
Global Const MPCMD_BORRAR = 5(0x05)' Borra un programa
Global Const MPCMD_CONFIG = 8(0x08)' Configuración puerto pantalla
Global Const MPCMD_GETPROG = 9(0x09)' Solicita un programa existente
Global Const MPCMD_SETHORA = 10(0x0A)' Establece la hora de la pantalla
Global Const MPCMD_GETHORA = 11(0x0B)' Solicita la hora de la pantalla
Global Const MPCMD_PASSWORD = 14(0x0E)' Envia el password a la pantalla
Global Const MPCMD_SETPASSW = 15(0x0F)' Establece nuevo password
Global Const MPCMD_SETTEMP = 16(0x10)' Establece offset temperatura
Global Const MPCMD_GETTEMP = 17(0x11)' Solicita temperatura
Global Const MPCMD_GETVER = 18(0x12)' Solicita versió de software
Global Const MPCMD_NGETDIR = 30(0x1E)' Solicita el directorio a la
                                       pantalla
Global Const MPCMD_NEJECUTAR = 31(0x1F)' Ejecuta un programa en la
                                         pantalla. Ver software Dynamic
                                         Plus
Global Const MPCMD_FASTEXEC = 39(0x27)' Ejecución instantánea de un
                                        programa
```

7.ANEXO B Protocolos soportados

```
Global Const MPPROT_RS232 = 0   ' Modo de transmisión puerto serie
Global Const MPPROT_TCPIP = 1   ' Modo de transmisión por Sockets TCP/IP
```

8.ANEXO C Constantes para programas

```
Global Const MPTIM_DIF_DIAS = &HA4
Global Const MPTIM_DIF_SEMAN = &HA5
Global Const MPTIM_DIF_MESES = &HA6
Global Const MPTIM_HORMIN = &HA7
Global Const MPTIM_TEMPER = &HA8
Global Const MPTIM_DIAV = &HA9
Global Const MPTIM_MESV = &HAA
Global Const MPEFE_FLASH = &HB0
Global Const MPEFE_NEGATIVO = &HB1
Global Const MPEFE_BORRADO = &HB2
Global Const MPEFE_ESPERA = &HB3
Global Const MPEFE_BEEP = &HB4
```

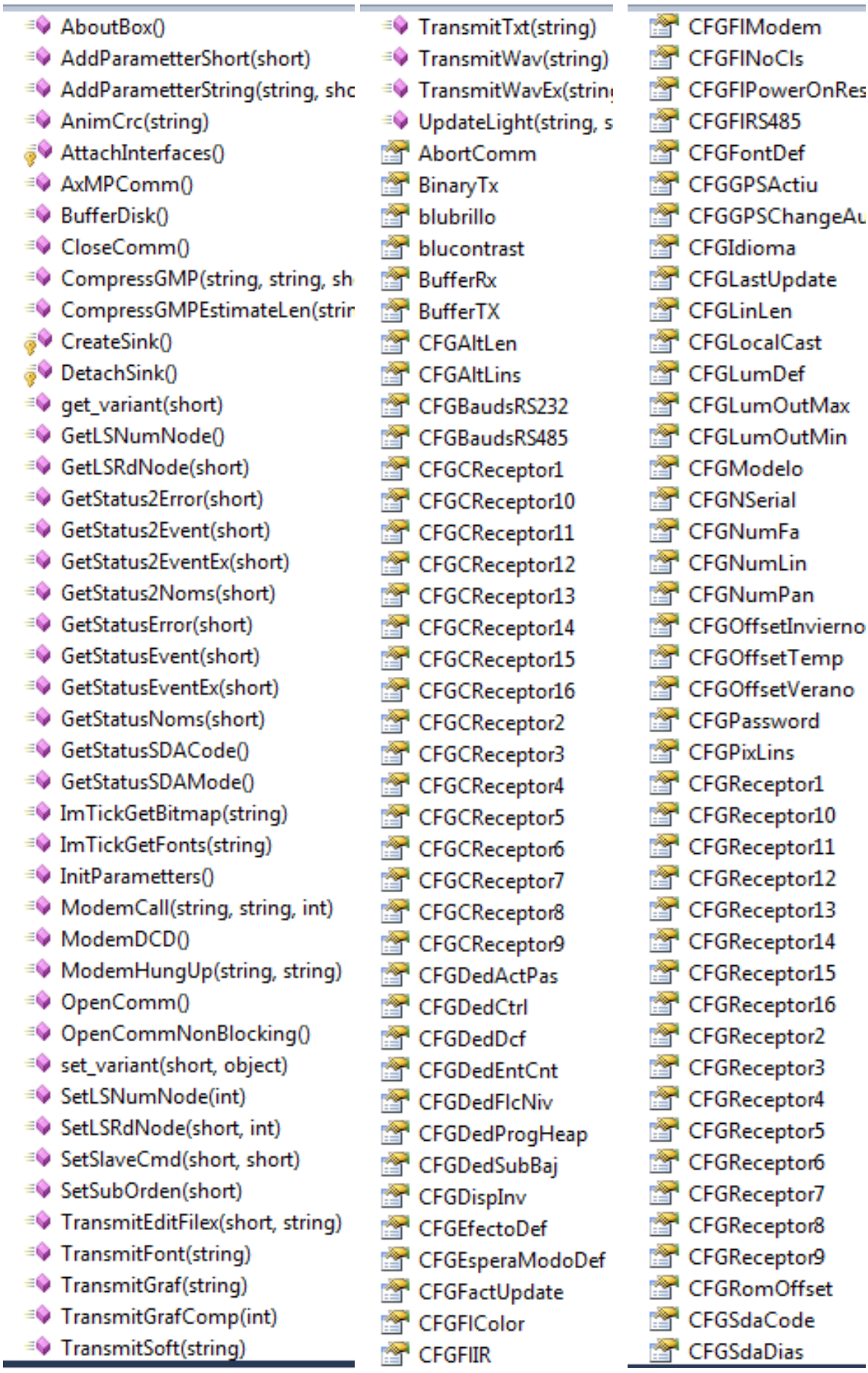
```
Global Const MPDAT_GROSOR = &HC0
Global Const MPDAT_TIPO_LETRA = &HC1
Global Const MPDAT_INVERSO = &HC2
Global Const MPDAT_NORMAL = &HC3
Global Const MPDAT_VEL_MODO = &HC4
Global Const MPDAT_ESP_MODO = &HC5
Global Const MPDAT_CICLOS = &HC6
Global Const MPDAT_LINIA = &HC7
Global Const MPDAT_PROGRAM = &HC8
Global Const MPDAT_SYNC = &HC9
Global Const MPDAT_NOSYNC = &HCA
Global Const MPDAT_NOCENTRO = &HCD
Global Const MPDAT_ANIM = &HCE
Global Const MPDAT_NOANIM = &HCF
Global Const MPDAT_LUMIN = &HD0
Global Const MPDAT_REM = &HD2
Global Const MPDAT_DIBUJO = &HD4
Global Const MPDAT_VENTANA = &HD3
Global Const MPDAT_MPBASIC = &HD5
Global Const MPTDAT_BLINK = &HA0
Global Const MPTDAT_COLOR = &HA1
Global Const MPTDAT_FONDO = &HA2
Global Const MPTDAT_GRAFICO = &HA3
Global Const MPMOD_CORRER = &HE0
Global Const MPMOD_CENTRO = &HE1
Global Const MPMOD_DISMINUIDO = &HE2
Global Const MPMOD_APILADO = &HE3
Global Const MPMOD_RODAR = &HE4
Global Const MPMOD_SUBE = &HE5
Global Const MPMOD_BAJA = &HE6
Global Const MPMOD_ROTACION1 = &HE7
Global Const MPMOD_ROTACION2 = &HE8
Global Const MPMOD_ROTACION3 = &HE9
Global Const MPMOD_ROTACION4 = &HEA
Global Const MPMOD_APARICION1 = &HEB
Global Const MPMOD_APARICION2 = &HEC
Global Const MPMOD_APARICION3 = &HED
Global Const MPMOD_APARICION4 = &HEE
Global Const MPMOD_NIEVE = &HEF
Global Const MPMOD_INMEDIATO = &HF0
Global Const MPMOD_DESLIZAR = &HF1
Global Const MPMOD_GIRAR = &HF2
Global Const MPMOD_BOLSA = &HF3
```

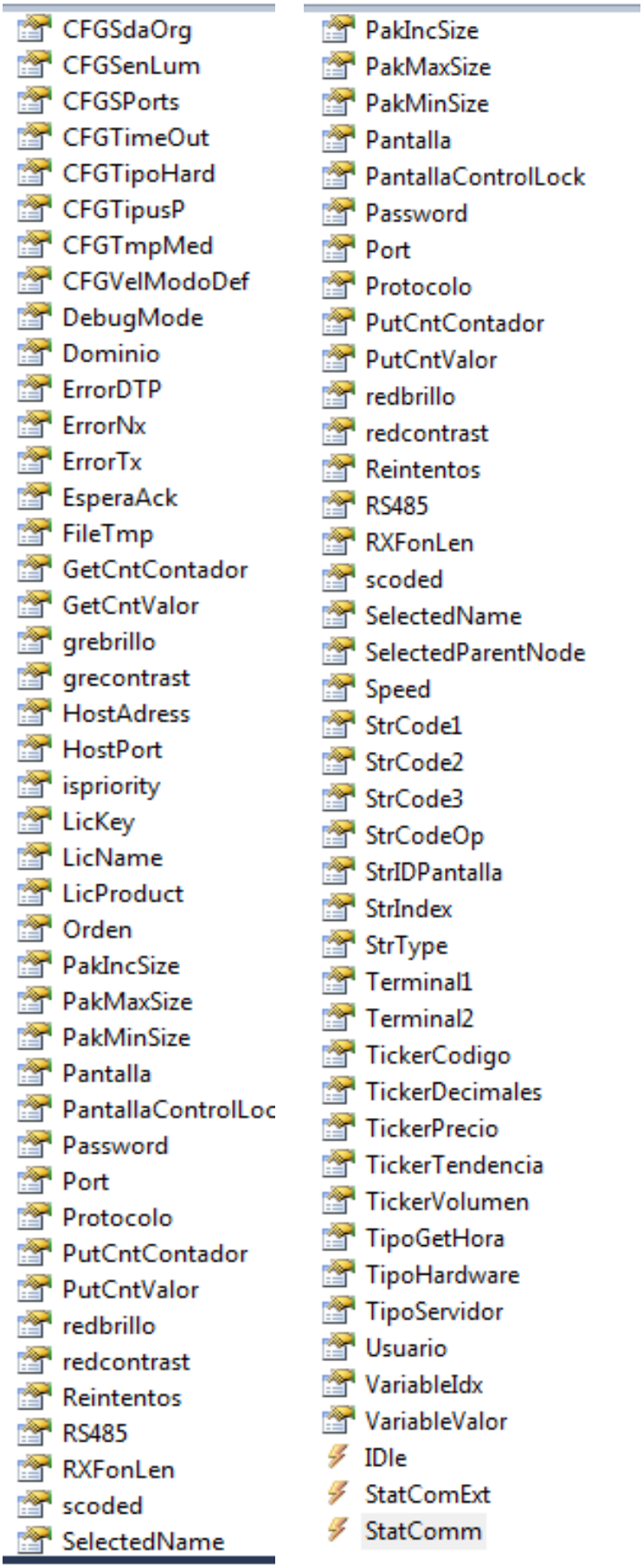
9.ANEXO D Lista de Métodos, Propiedades y eventos

La lista siguiente es exhaustiva e incluye las propiedaes, métodos y eventos utilizados exclusivamente para la programación internadel visualizador en fábrica.

Estos no aparecen en este manual que está orientado al usuario final.

Se recomienda de no modificar su valor por defecto y utilizar solamente los que se han descrito en las páginas anteriores de este manual.


AboutBox()
AddParametterShort(short)
AddParametterString(string, shc
AnimCrc(string)
AttachInterfaces()
AxMPComm()
BufferDisk()
CloseComm()
CompressGMP(string, string, sh
CompressGMPEstimateLen(strir
CreateSink()
DetachSink()
get_variant(short)
GetLSNumNode()
GetLSRdNode(short)
GetStatus2Error(short)
GetStatus2Event(short)
GetStatus2EventEx(short)
GetStatus2Noms(short)
GetStatusError(short)
GetStatusEvent(short)
GetStatusEventEx(short)
GetStatusNoms(short)
GetStatusSDACode()
GetStatusSDAMode()
ImTickGetBitmap(string)
ImTickGetFonts(string)
InitParametters()
ModemCall(string, string, int)
ModemDCD()
ModemHungUp(string, string)
OpenComm()
OpenCommNonBlocking()
set_variant(short, object)
SetLSNumNode(int)
SetLSRdNode(short, int)
SetSlaveCmd(short, short)
SetSubOrden(short)
TransmitEditFilex(short, string)
TransmitFont(string)
TransmitGraf(string)
TransmitGrafComp(int)
TransmitSoft(string)
TransmitTxt(string)
TransmitWav(string)
TransmitWavEx(strin
UpdateLight(string, s
AbortComm
BinaryTx
blubrillo
blucontrast
BufferRx
BufferTX
CFGAltLen
CFGAltLins
CFGBaudsRS232
CFGBaudsRS485
CFGCReceptor1
CFGCReceptor10
CFGCReceptor11
CFGCReceptor12
CFGCReceptor13
CFGCReceptor14
CFGCReceptor15
CFGCReceptor16
CFGCReceptor2
CFGCReceptor3
CFGCReceptor4
CFGCReceptor5
CFGCReceptor6
CFGCReceptor7
CFGCReceptor8
CFGCReceptor9
CFGDedActPas
CFGDedCtrl
CFGDedDcf
CFGDedEntCnt
CFGDedFlcNiv
CFGDedProgHeap
CFGDedSubBaj
CFGDispInv
CFGEfectoDef
CFGEsperaModoDef
CFGFactUpdate
CFGFlColor
CFGFIIR
CFGFlModem
CFGFlNoCls
CFGFlPowerOnRes
CFGFlRS485
CFGFontDef
CFGGPSActiu
CFGGPSChangeAu
CFGIdioma
CFGLastUpdate
CFGLinLen
CFGLocalCast
CFGLumDef
CFGLumOutMax
CFGLumOutMin
CFGModelo
CFGNSerial
CFGNumFa
CFGNumLin
CFGNumPan
CFGOffsetInvierno
CFGOffsetTemp
CFGOffsetVerano
CFGPassword
CFGPixLins
CFGReceptor1
CFGReceptor10
CFGReceptor11
CFGReceptor12
CFGReceptor13
CFGReceptor14
CFGReceptor15
CFGReceptor16
CFGReceptor2
CFGReceptor3
CFGReceptor4
CFGReceptor5
CFGReceptor6
CFGReceptor7
CFGReceptor8
CFGReceptor9
CFGRomOffset
CFGSdaCode
CFGSdaDias


CFGSdaOrg
CFGSenLum
CFGSPorts
CFGTimeOut
CFGTipoHard
CFGTipusP
CFGTmpMed
CFGVelModoDef
DebugMode
Dominio
ErrorDTP
ErrorNx
ErrorTx
EsperaAck
FileTmp
GetCntContador
GetCntValor
grebrillo
grecontrast
HostAdress
HostPort
ispriority
LicKey
LicName
LicProduct
Orden
PakIncSize
PakMaxSize
PakMinSize
Pantalla
PantallaControlLoc
Password
Port
Protocolo
PutCntContador
PutCntValor
redbrillo
redcontrast
Reintentos
RS485
RXFonLen
scoded
SelectedName
PakIncSize
PakMaxSize
PakMinSize
Pantalla
PantallaControlLock
Password
Port
Protocolo
PutCntContador
PutCntValor
redbrillo
redcontrast
Reintentos
RS485
RXFonLen
scoded
SelectedName
SelectedParentNode
Speed
StrCode1
StrCode2
StrCode3
StrCodeOp
StrIDPantalla
StrIndex
StrType
Terminal1
Terminal2
TickerCodigo
TickerDecimales
TickerPrecio
TickerTendencia
TickerVolumen
TipoGetHora
TipoHardware
TipoServidor
Usuario
VariableIdx
VariableValor
IDle
StatComExt
StatComm

10. Utilización de MPCOMM.OCX en una aplicación .NET

A continuación se explica el mejor método para utilizar el componente MPComm.OCX en una Aplicación.net

Todos los ejemplos se presentan en lenguaje C#.

Crear los Wrappers COM (Component Object Model) para OCX

El OCX no se puede utilizar directamente en la aplicación.

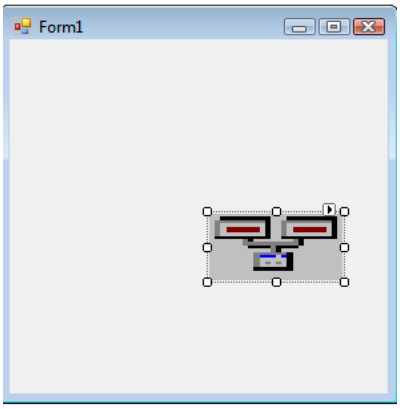
Es necesario crear una serie de DLL (wrappers).

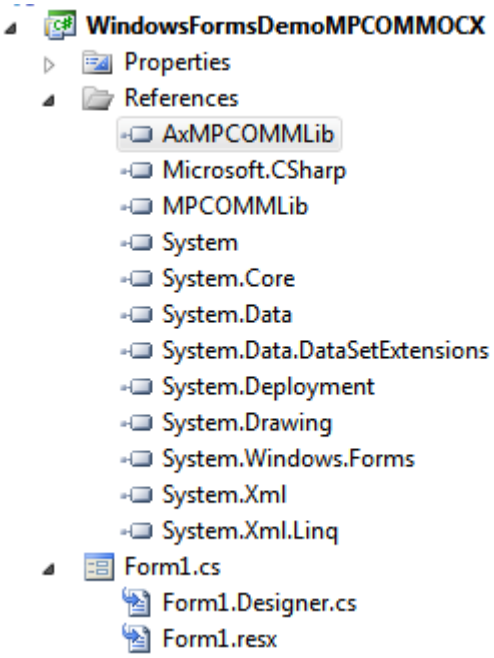
El modo más simple es (para Visual Studio 2010 y lenguage C #):

- Crear un formulario, clicar sobre la caja de herramientas y seleccionar « Escoger elementos ».
- Clicar sobre la pestaña “Componentes COM”.
- Localizar su control ActiveX (MPCOMM: OCX), marcar y clicar sobre « OK ».
- Ahora deberia aparecer sobre la caja de herramientas; arrastrarla y depositarla sobre vuestro formulario. Darle un nombre (por ejemplo: AxMPCOMM1)

NOTA: Comprobar que en la sección « referencias » de su proyecto se han añadido automáticamente dos elementos (wrappers):

. AxMPCOMMLib
. MPCOMMLib





11. Programming codes

Using the component

Configure Component

```
            axMPComm1.LicProduct = "MPCOMM";
            axMPComm1.LicName = "PIXELMP";
            axMPComm1.LicKey = "431A2CF1683AAFAC";
            axMPComm1.Terminal2 = 8;
            axMPComm1.Terminal1 = 4;
            axMPComm1.DebugMode = true;
```

Open Communications (in this case RS232):

```
            // Open RS232
            axMPComm1.Port = "COM1:";
            axMPComm1.Speed = "9600";
            axMPComm1.Protocolo = 0;     // 0 -> RS232/485, 1 -> TCP/
IP
            axMPComm1.RS485 = false;
```

Prepare Command to Send

```
            // Open Comm
            bool ret =axMPComm1.OpenComm();

            // RS232 successfully open
            if (ret)
            {
                // Target device
                axMPComm1.Pantalla = 1;   // Device with ID 1

                // Mode 'Scroll' + 'MP test'
                      axMPComm1.BufferTX = (char) 0xE0 + "MP Test";
                      axMPComm1.Orden = 0x27;   // Send

                        // Receive (if any)
                        String s = axMPComm1.BufferRx;  // Get answer
(if any)

                        // Error
                        int error = axMPComm1.ErrorNx;  // Get error
number

                        // Close Comm
                        axMPComm1.CloseComm();
                    }
```

Example 1: serial network

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsDemoMPCOMMOCX
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // Configure component
            axMPComm1.LicProduct = "MPCOMM";
            axMPComm1.LicName = "PIXELMP";
            axMPComm1.LicKey = "431A2CF1683AAFAC";
            axMPComm1.Terminal2 = 8;
            axMPComm1.Terminal1 = 4;
            axMPComm1.DebugMode = false;

            // Open RS232
            axMPComm1.Port = "COM8:";
            axMPComm1.Speed = "9600";
            axMPComm1.Protocolo = 0;      // 0 -> RS232/485, 1 -> TCP/IP
            axMPComm1.RS485 = false;

            // Open Comm
            bool ret =axMPComm1.OpenComm();

            // RS232 successfully open
            if (ret)
            {
                // Target device
                axMPComm1.Pantalla = 1;         // Device with ID 1
                axMPComm1.BufferTX = (char) 0xE0 + "MP Test"; // Mode
'Scroll' + 'MP test'
                axMPComm1.Orden = 0x27;           // Send
                // Receive (if any)
                String s = axMPComm1.BufferRx;    // Get answer (if any)
             // Error
              int error = axMPComm1.ErrorNx;              // Get error number
             // Close Comm
             axMPComm1.CloseComm();
          }
       }
   }
```

Example 2: serial and Ethernet networks

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsDemoMPCOMMOCX
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      public void Test()
      {
         // Configure component
         Commom_Ini();


         // Test RS232_485
         RS232_485Version();

         // Test Ethernet TCP_IP
         TCP_IPVersion();
      }

      public void Commom_Ini()
      {
         // Configure component
         axMPComm1.LicProduct = "MPCOMM";
         axMPComm1.LicName = "PIXELMP";
         axMPComm1.LicKey = "431A2CF1683AAFAC";
         axMPComm1.Terminal2 = 8;
         axMPComm1.Terminal1 = 4;
         axMPComm1.DebugMode = false;
      }

      public void Common_Send(String s1)
      {
         // Open Comm
         bool ret = axMPComm1.OpenComm();

         // RS232 successfully open
         if (ret)
         {
```

```
            // Target device

            axMPComm1.Pantalla = 1;                    // Device with ID 1
            axMPComm1.BufferTX = (char)0x9A + 0xE4 + s1;          // Mode 'Scroll' + 'MP test'
            axMPComm1.Orden = 0x27;                    // Send

            // Receive (if any)
            String s = axMPComm1.BufferRx;             // Get answer (if any)

            // Error
            int error = axMPComm1.ErrorNx;             // Get error number

            // Close Comm
            axMPComm1.CloseComm();
        }
    }

    public void RS232_485Version()
    {
        // Open RS232
        axMPComm1.Port = "COM2:";
        axMPComm1.Speed = "9600";
        axMPComm1.Protocolo = 0;            // 0 -> RS232/485, 1 -> TCP/IP
        axMPComm1.RS485 = false;

        // Open Comm
        Common_Send("MP Test RS232485");
    }

    public void TCP_IPVersion()
    {
        // Open RS232
        axMPComm1.TipoServidor = 1;
        axMPComm1.HostPort = 53;
        axMPComm1.Usuario = "admin";
        axMPComm1.Password = "security";
        axMPComm1.HostAdress = "192.168.1.44";

        axMPComm1.Protocolo = 1;


        // Open Comm
        Common_Send("MP Test TCP IP");
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        Test();
      Close();
    }
  }
}
```